Use Case Study

18/June/2004

Bill Tanenbaum

US-CMS/Fermilab

Outline

- Introduction to DST's and Reconstructed Objects
- Restatement of Use Case
- "Solution" of Use Case

Stages of CMS Production

- **SimHits (Simulation)**
  - Produced by OSCAR (using GEANT4) OR reformatted from Zebra file (GEANT3) by ORCA (writeHits)
- **Digis (Digitization)**
  - Produced by ORCA (writeAllDigis) from SimHits
- **DST (Reconstruction)**
  - Produced by ORCA (writeDST) from Digis
  - About 50 reconstruction algorithms for various types of objects (e.g. tracks, vertices, Jets, etc.)
  - Reconstructed analysis objects stored in collections.
- **Analysis**
  - Not (yet?) part of production process

Reconstructed Objects

- **Stored persistently in DST**
- **Each object type must inherit from RecObj**
  - Provides persistency capability
- **Objects are hierarchical**
  - Higher level objects are constructed from lower level objects
  - Examples: Jets are reconstructed from Tracks, Vertices are reconstructed from Tracks, etc.
  - At the lowest level, all objects inherit from raw data (Digis)

Collections of Analysis Objects

- **Defined by:**
  - Type of object (e.g. Track, Vertex, Jet)
  - Name of algorithm (must be unique)
  - Version of algorithm
  - Parameter Set
    - Data independent parameters
    - Does not include parameters of components
  - Component Set
    - Dependencies on other (lower level) objects
  - Calibration Set
    - Data dependent parameters
- **All these define the RecConfig**
  - The same RecConfig gives reproducible results

Collections (cont.)

- **One (documented) interface for retrieving collections**
  - RecCollection
- **Used to iterate over homogeneous collections**
  - For analysis or for reconstructing higher level objects
- **Collections are homogeneous**
  - Within an event
  - For all events, reproducibly
- **Reconstruction is expensive**
  - Do it only on demand
  - Cache results
  - Make persistent if desired

RecQuery

- **User friendly way to define RecConfig**
  - Only the algorithm name is required
    - Everything else is taken from (algorithm specific) default configuration
    - Only the parameters, components, or calibrations that need to be changed need be specified
    - Parameters can be specified in query itself or in .orcarc.
    - Query arguments override the .orcarc settings, which override defaults.
  - Errors (e.g. undefined algorithm name, parameter name, etc.) found only at runtime (exception will be thrown)
- **More information about Reconstructed Objects**
  - See Norbert Neumeister's DST tutorial
  - Norbert's tutorial shows how to define algorithms, parameter sets, etc.

Restatement of Use Case

Actor: Physicist

Goal: As part of debugging an algorithm, print some parameters of each reconstructed track found by a specific algorithm.

Trigger: My code has activated by the presence of a new event that needs my attention.

In the current event, I will ask for all those tracks found by algorithm named *X* version *n*, with completely unique and unambiguous configuration *Y*. There must be either 0 or 1 collections of tracks associated with this run of the algorithm.

If this version of this algorithm has not previously been run with this configuration, I will print a message saying it has not been run.

If this algorithm has been run, I will iterate through all those tracks, printing to standard output the transverse momentum of the track.

Caveats on "Solution"

- **Selection of Algorithm version**
  - Currently, the DST's have only one version of each algorithm
  - So, specifying the version is optional. It defaults to the correct version
  - Bug (?) found: The algorithm version is effectively ignored on query. I fixed the bug so that the version is checked. This fix (more or less) should become official, pending approval of the owners of the code.
  - Since there is only one version defined, specifying a different version (with the fix in place) causes a run time exception because of a non-existent version of the algorithm. Arguably, this is proper behavior.
- **Selection of Configuration**
  - Includes Parameter Set, Component Set, and Calibration Set
  - Currently, a given DST use only one configuration for each algorithm
  - So the default can be used. I did, for simplicity.
  - The configuration is properly checked for a match.

Caveats on "Solution" (cont.)

- **Selection of Event**
  - I didn't bother to trigger on one specific event. For simplicity, the printout is triggered for every event.
  - It's simple to add the event selection criteria of your choice.
- **Skipping an event that did not use the chosen algorithm**
  - Not currently supported by the documented interface, as far as I can tell
  - Reconstruction on demand (see later) will cause the query to be done.
  - Need an enhancement to enable this.
  - I will request it, if it is not already there.

Comments on the Solution

- **Most of code is standard “template” code**
  - Only the analysis() method is specific to this use case. It is automatically called once per event by the framework.
  - However, it is important to write the BuildFile so that the correct shared libraries are specified.
  - Solution similar to existing ExDSTStatistics program.
- **The collection is initialized only once, at beginning of run**
  - The collection is a lazy observer of dispatched events.
  - When the collection is accessed, it automatically updates itself from the new event if a new event has been dispatched since the previous access. The objects from the previous event, if any, are first cleared.
  - The configuration is guaranteed consistent across events.
  - If the collection is not accessed for a particular event, it is never updated for that event (reconstruction on demand).

The Guts of the Solution

```
void UseCase::analysis(G3EventProxy* ev)  {
   if (theTkCollection == 0) {
      theTkCollection = new RecCollection<TTrack>(RecQuery("CombinatorialTrackFinder“, “0.0”));
   }  // Note:  configuration arguments are defaulted.  Version (“0.0”) could also be defaulted.
  cout << "Found " << theTkCollection->size() << " Tracker tracks." << endl;
  GlobalVector p;
  for(RecCollection<TTrack>::const_iterator it=theTkCollection->begin();
    it!=theTkCollection->end(); it++) {
      p = (*it)->momentumAtVertex();
      cout << "p="   << setw(8) << setprecision(3) << p.mag()  // total momentum
     << " GeV, pt=“ << setw(8) << setprecision(4) << p.perp()  // transverse momentum
     << " GeV, theta=“ << setw(8) << setprecision(4) << p.theta() // theta of momentum
     << " rad, phi=" << setw(8) << setprecision(4) << p.phi() // phi of momentum
     << " rad, eta="  << setw(8) << setprecision(4) << p.eta()  << endl; // pseudorapidity
    }
  }
```